Martín López de Ipiña:

Refactor1 (capítulo2):

La función ModifyBet tiene demasiadas líneas (muchas más que 15) y hace demasiadas cosas, hay que dividirla en varias funciones:

```
public Bet modifyBet (float betMoney, int betNumber, String dni) throws BetDoesntExist,
UserDoesntExist {

   if(dni == null){
      System.err.println(">> DataAccess: modifyBet => error UserDoesntExist: error, dni nulo");
      throw new UserDoesntExist("El usuario introducido no es correcto");
   }
   if(betNumber < 0){
      System.err.println(">> DataAccess: modifyBet => error BetDoesntExist: error, identificador
negativo");
      throw new BetDoesntExist("No existe la apuesta a modificar");
   }

   Bet bet = db.find(Bet.class, betNumber);
   User user = db.find(User.class, dni);

   if (user == null) {
      System.err.println(">> DataAccess: modifyBet => error UserDoesntExist: No existe un usuario con
este DNI en la base de datos, dni="+dni);
      throw new UserDoesntExist("No existe un usuario con este DNI en la base de datos, dni="+dni);
   }
   if (bet == null) {
      System.err.println(">> DataAccess: modifyBet => error BetDoesntExist: No existe la apuesta ha
modificar");
      throw new BetDoesntExist("No existe la apuesta a modificar");
   }

   double betMoneyBefore = bet.getBetMoney();
   double betMoneyAfter = betMoney + betMoneyBefore;
   float userMoney = user.getSaldo();

   if(userMoney >= betMoney && betMoneyAfter > 0){

      user.setSaldo(userMoney - betMoney);
      bet.setBetMoney((float)betMoneyAfter);

      db.getTransaction().begin();
      db.persist(user);
      db.persist(bet);
      db.getTransaction().commit();
   }
```

He partido la función en trozos pequeños para que se entienda mejor el funcionamiento del programa:

```
public Bet modifyBet (float betMoney, int betNumber, String dni) throws BetDoesntExist,
UserDoesntExist {

   checkIfDNIAndBetNumberRight(betNumber, dni);

   Bet bet = db.find(Bet.class, betNumber);
   User user = db.find(User.class, dni);

   checkIfUserAndBetInDB(user, bet);

   modifyBetIfPossible(betMoney, bet, user);
   return bet;
}


private void checkIfDNIAndBetNumberRight(int betNumber, String dni) throws
UserDoesntExist, BetDoesntExist {
   checkIfDNIRight(dni);
   checkIfBetNumberRight(betNumber);
}


private void checkIfBetNumberRight(int betNumber) throws BetDoesntExist {
   if(betNumber < 0){
      System.err.println(">> DataAccess: modifyBet => error BetDoesntExist: error, identificador
negativo");
      throw new BetDoesntExist("No existe la apuesta a modificar");
   }
}


private void checkIfDNIRight(String dni) throws UserDoesntExist {
   if(dni == null){
      System.err.println(">> DataAccess: modifyBet => error UserDoesntExist: error, dni nulo");
      throw new UserDoesntExist("El usuario introducido no es correcto");
   }
}


private void checkIfUserAndBetInDB(User user, Bet bet) throws UserDoesntExist,
BetDoesntExist {
   checkIfUserInDB(user);
   checkIfBetInDB(bet);
}

private void checkIfUserInDB(User user) throws UserDoesntExist {
   if (user == null) {
      System.err.println(">> DataAccess: modifyBet => error UserDoesntExist: No existe un
usuario con este DNI en la base de datos");
      throw new UserDoesntExist("No existe un usuario con este DNI en la base de datos");
```

```
    }
}

private void checkIfBetInDB(Bet bet) throws BetDoesntExist {
    if (bet == null) {
        System.err.println(">> DataAccess: modifyBet => error BetDoesntExist: No existe la apuesta
ha modificar");
        throw new BetDoesntExist("No existe la apuesta a modificar");
    }
}

private void modifyBetIfPossible(float betMoney, Bet bet, User user) {
    double betMoneyBefore = bet.getBetMoney();
    double betMoneyAfter = betMoney + betMoneyBefore;
    float userMoney = user.getSaldo();

    if(betPossible(betMoney, userMoney, betMoneyAfter)){

        modifyBetValues(betMoney, bet, user, userMoney, (float) betMoneyAfter);

        persistModifiedBetAndUserIntoDB(bet, user);
    }
}

private boolean betPossible(float betMoney, float userMoney, double betMoneyAfter) {
    return userMoney >= betMoney && betMoneyAfter > 0;
}

private void modifyBetValues(float betMoney, Bet bet, User user, float userMoney, float
betMoneyAfter) {
    user.setSaldo(userMoney - betMoney);
    bet.setBetMoney(betMoney);
}

private void persistModifiedBetAndUserIntoDB(Bet bet, User user) {
    db.getTransaction().begin();
    db.persist(user);
    db.persist(bet);
    db.getTransaction().commit();
}
```

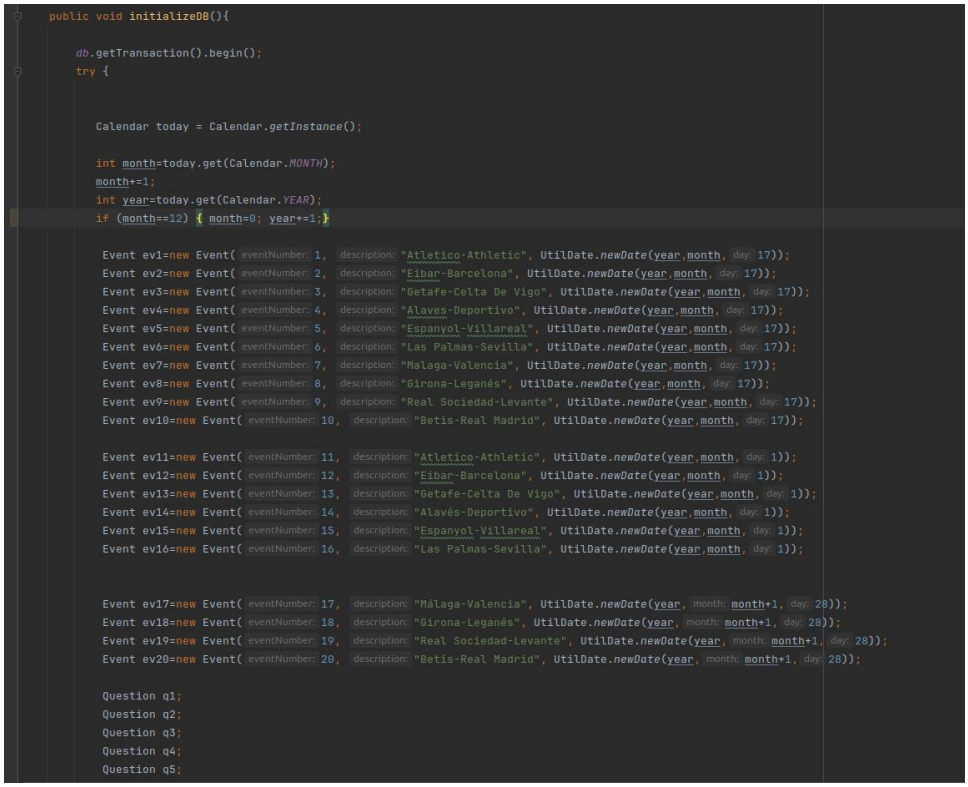
Ahora la función se divide en 3 subfunciones que a la vez se dividen en 2 subfunciones:

- checkIfDNIAndBetNumberRight
- checkIfUserAndBetInDB
- modifyBetIfPossible

De esta manera cada función hace una cosa específica que es la que su nombre indica.

Refactor2:(Capítulo4):

Como se puede ver en la función de initializeDB se repite el código muchas veces:



```
public void initializeDB(){

    db.getTransaction().begin();
    try {


      Calendar today = Calendar.getInstance();

      int month=today.get(Calendar.MONTH);
      month+=1;
      int year=today.get(Calendar.YEAR);
      if (month==12) { month=0; year+=1;}

       Event ev1=new Event( eventNumber: 1,  description: "Atletico-Athletic", UtilDate.newDate(year,month, day: 17));
       Event ev2=new Event( eventNumber: 2,  description: "Eibar-Barcelona", UtilDate.newDate(year,month, day: 17));
       Event ev3=new Event( eventNumber: 3,  description: "Getafe-Celta De Vigo", UtilDate.newDate(year,month, day: 17));
       Event ev4=new Event( eventNumber: 4,  description: "Alaves-Deportivo", UtilDate.newDate(year,month, day: 17));
       Event ev5=new Event( eventNumber: 5,  description: "Espanyol-Villareal", UtilDate.newDate(year,month, day: 17));
       Event ev6=new Event( eventNumber: 6,  description: "Las Palmas-Sevilla", UtilDate.newDate(year,month, day: 17));
       Event ev7=new Event( eventNumber: 7,  description: "Malaga-Valencia", UtilDate.newDate(year,month, day: 17));
       Event ev8=new Event( eventNumber: 8,  description: "Girona-Leganés", UtilDate.newDate(year,month, day: 17));
       Event ev9=new Event( eventNumber: 9,  description: "Real Sociedad-Levante", UtilDate.newDate(year,month, day: 17));
       Event ev10=new Event( eventNumber: 10,  description: "Betis-Real Madrid", UtilDate.newDate(year,month, day: 17));

       Event ev11=new Event( eventNumber: 11,  description: "Atletico-Athletic", UtilDate.newDate(year,month, day: 1));
       Event ev12=new Event( eventNumber: 12,  description: "Eibar-Barcelona", UtilDate.newDate(year,month, day: 1));
       Event ev13=new Event( eventNumber: 13,  description: "Getafe-Celta De Vigo", UtilDate.newDate(year,month, day: 1));
       Event ev14=new Event( eventNumber: 14,  description: "Alavés-Deportivo", UtilDate.newDate(year,month, day: 1));
       Event ev15=new Event( eventNumber: 15,  description: "Espanyol-Villareal", UtilDate.newDate(year,month, day: 1));
       Event ev16=new Event( eventNumber: 16,  description: "Las Palmas-Sevilla", UtilDate.newDate(year,month, day: 1));


       Event ev17=new Event( eventNumber: 17,  description: "Málaga-Valencia", UtilDate.newDate(year, month: month+1, day: 28));
       Event ev18=new Event( eventNumber: 18,  description: "Girona-Leganés", UtilDate.newDate(year, month: month+1, day: 28));
       Event ev19=new Event( eventNumber: 19,  description: "Real Sociedad-Levante", UtilDate.newDate(year, month: month+1, day: 28));
       Event ev20=new Event( eventNumber: 20,  description: "Betis-Real Madrid", UtilDate.newDate(year, month: month+1, day: 28));

       Question q1;
       Question q2;
       Question q3;
       Question q4;
       Question q5;
```

```
        Question q6;

        if (Locale.getDefault().equals(new Locale( language: "es"))) {
            q1=ev1.addQuestion( question: "¿Quién ganará el partido?", betMinimum: 1);
            q2=ev1.addQuestion( question: "¿Quién meterá el primer gol?", betMinimum: 2);
            q3=ev11.addQuestion( question: "¿Quién ganará el partido?", betMinimum: 1);
            q4=ev11.addQuestion( question: "¿Cuántos goles se marcarán?", betMinimum: 2);
            q5=ev17.addQuestion( question: "¿Quién ganará el partido?", betMinimum: 1);
            q6=ev17.addQuestion( question: "¿Habrá goles en la primera parte?", betMinimum: 2);
        }
        else if (Locale.getDefault().equals(new Locale( language: "en"))) {
            q1=ev1.addQuestion( question: "Who will win the match?", betMinimum: 1);
            q2=ev1.addQuestion( question: "Who will score first?", betMinimum: 2);
            q3=ev11.addQuestion( question: "Who will win the match?", betMinimum: 1);
            q4=ev11.addQuestion( question: "How many goals will be scored in the match?", betMinimum: 2);
            q5=ev17.addQuestion( question: "Who will win the match?", betMinimum: 1);
            q6=ev17.addQuestion( question: "Will there be goals in the first half?", betMinimum: 2);
        }
        else {
            q1=ev1.addQuestion( question: "Zeinek irabaziko du partidua?", betMinimum: 1);
            q2=ev1.addQuestion( question: "Zeinek sartuko du lehenengo gola?", betMinimum: 2);
            q3=ev11.addQuestion( question: "Zeinek irabaziko du partidua?", betMinimum: 1);
            q4=ev11.addQuestion( question: "Zenbat gol sartuko dira?", betMinimum: 2);
            q5=ev17.addQuestion( question: "Zeinek irabaziko du partidua?", betMinimum: 1);
            q6=ev17.addQuestion( question: "Golak sartuko dira lehenengo zatian?", betMinimum: 2);

        }

        q1.addForecast( description: "Athletic",  gain: 1.5f ,q1);
        q1.addForecast( description: "Atlético",  gain: 1.4f ,q1);
        q2.addForecast( description: "Athletic",  gain: 1.8f ,q2);
        q2.addForecast( description: "Atlético",  gain: 1.2f ,q2);

        db.persist(q1);
        db.persist(q2);
        db.persist(q3);
        db.persist(q4);
        db.persist(q5);
        db.persist(q6);


        db.persist(ev1);
```

```
            db.persist(ev1);
            db.persist(ev2);
            db.persist(ev3);
            db.persist(ev4);
            db.persist(ev5);
            db.persist(ev6);
            db.persist(ev7);
            db.persist(ev8);
            db.persist(ev9);
            db.persist(ev10);
            db.persist(ev11);
            db.persist(ev12);
            db.persist(ev13);
            db.persist(ev14);
            db.persist(ev15);
            db.persist(ev16);
            db.persist(ev17);
            db.persist(ev18);
            db.persist(ev19);
            db.persist(ev20);


            db.getTransaction().commit();
            System.out.println("Db initialized");
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
```

- Se llama al constructor de Event 20 veces
- Se declaran 6 questions y por cada idioma se ponen individualmente, repitiendo el código 3 veces
- Al hacer persist de los eventos y questions se hacen individualmente, repitiéndolo 26 veces

He almacenado las variables en listas y refactorizado el código para evitar estas repeticiones:

```
public void initializeDB(){

  db.getTransaction().begin();
   try {


    Calendar today = Calendar.getInstance();

    int month=today.get(Calendar.MONTH);
     month+=1;
     int year=today.get(Calendar.YEAR);
     if (month==12) { month=0; year+=1;}

    List<String> eventNames = new ArrayList(Arrays.asList("Atletico-Athletic","Eibar-Barcelona","Getafe-
Celta","Alaves-Deportivo","Espanyol-Villareal",
        "Las Palmas-Sevilla","Malaga-Valencia","Girona-Leganés","Real Sociedad-Levante","Betis-Real
Madrid", "Atletico-Athletic", "Eibar-Barcelona",
        "Getafe-Celta", "Alaves-Deportivo", "Espanyol-Villareal", "Las Palmas-Sevilla", "Málaga-Valencia",
"Girona-Leganés", "Real Sociedad-Levante", "Betis-Real Madrid"));

    List<Event> events = new ArrayList<>();
    for(int i = 1; i < eventNames.size()+1; i++){
      Event event = null;
      if(i <= 10){
        event = new Event(i, eventNames.get(i), UtilDate.newDate(year, month, 17));
      }
      if(i >= 11 && i <= 16){
        event = new Event(i, eventNames.get(i), UtilDate.newDate(year, month, 1));
      }
      if(i >= 17 && i <= 20){
        event = new Event(i, eventNames.get(i), UtilDate.newDate(year, month+1, 28));
      }
      events.add(event);
    }

    List<String>[] questionNames = new ArrayList[]{
        new ArrayList(Arrays.asList("¿Quién ganará el partido?", "¿Quién meterá el primer gol?", "¿Cuántos
goles se marcarán?", "¿Habrá goles en la primera parte?")),
        new ArrayList(Arrays.asList("Who will win the match?", "Who will score first?", "How many goals
will be scored in the match?", "Will there be goals in the first half?")),
        new ArrayList(Arrays.asList("Zeinek irabaziko du partidua?", "Zeinek sartuko du lehenengo gola?",
"Zenbat gol sartuko dira?", "Golak sartuko dira lehenengo zatian?"))
    };

    List<Question> questions = new ArrayList<>();
    switch (Locale.getDefault().toString()){
      case "es_ES":
        questions = createQuestion(questionNames[0], events);
      case "en_US":
        questions = createQuestion(questionNames[1], events);
      case "eu_ES":
        questions = createQuestion(questionNames[2], events);
      default:
        questions = createQuestion(questionNames[0], events);
```

```
        }

        questions.get(0).addForecast("Athletic", 1.5f ,questions.get(0));
        questions.get(0).addForecast("Atlético", 1.4f ,questions.get(0));
        questions.get(1).addForecast("Athletic", 1.8f ,questions.get(1));
        questions.get(1).addForecast("Atlético", 1.2f ,questions.get(1));

        for(Question q: questions){
          db.persist(q);
        }
        for(Event ev: events){
          db.persist(ev);
        }

        db.getTransaction().commit();
        System.out.println("Db initialized");
      }
      catch (Exception e){
        e.printStackTrace();
      }
    }


    private List<Question> createQuestion(List<String> questionNames, List<Event> events){
      List<Question> questions = new ArrayList<>();

      questions.add(events.get(0).addQuestion(questionNames.get(0), 1));
      questions.add(events.get(0).addQuestion(questionNames.get(1), 2));
      questions.add(events.get(10).addQuestion(questionNames.get(0), 1));
      questions.add(events.get(10).addQuestion(questionNames.get(2), 2));
      questions.add(events.get(16).addQuestion(questionNames.get(0), 1));
      questions.add(events.get(16).addQuestion(questionNames.get(3), 2));

      return questions;
    }
```

Refactor3(Capítulo3):

Siguiendo la función initializeDB que he refactorizado en el refactor anterior, hay demasiados puntos de ramificación, por lo que hay que extraer algunos bucles y condicionales fuera:

```
public void initializeDB(){

  db.getTransaction().begin();
   try {


    Calendar today = Calendar.getInstance();

    int month=today.get(Calendar.MONTH);
     month+=1;
```

```
        int year=today.get(Calendar.YEAR);
        if (month==12) { month=0; year+=1;}

        List<String> eventNames = new ArrayList(Arrays.asList("Atletico-Athletic","Eibar-
Barcelona","Getafe-Celta","Alaves-Deportivo","Espanyol-Villareal",
                "Las Palmas-Sevilla","Malaga-Valencia","Girona-Leganés","Real Sociedad-Levante","Betis-
Real Madrid", "Atletico-Athletic", "Eibar-Barcelona",
                "Getafe-Celta", "Alaves-Deportivo", "Espanyol-Villareal", "Las Palmas-Sevilla", "Málaga-
Valencia", "Girona-Leganés", "Real Sociedad-Levante", "Betis-Real Madrid"));

        List<Event> events = createEvents(eventNames, year, month);

        List<String>[] questionNames = new ArrayList[]{
                new ArrayList(Arrays.asList("¿Quién ganará el partido?", "¿Quién meterá el primer gol?",
"¿Cuántos goles se marcarán?", "¿Habrá goles en la primera parte?")),
                new ArrayList(Arrays.asList("Who will win the match?", "Who will score first?", "How
many goals will be scored in the match?", "Will there be goals in the first half?")),
                new ArrayList(Arrays.asList("Zeinek irabaziko du partidua?", "Zeinek sartuko du
lehenengo gola?", "Zenbat gol sartuko dira?", "Golak sartuko dira lehenengo zatian?"))
        };

        List<Question> questions = createQuestionsByLanguaje(questionNames, events);

       questions.get(0).addForecast("Athletic", 1.5f ,questions.get(0));
        questions.get(0).addForecast("Atlético", 1.4f ,questions.get(0));
        questions.get(1).addForecast("Athletic", 1.8f ,questions.get(1));
        questions.get(1).addForecast("Atlético", 1.2f ,questions.get(1));

        persistQuestions(questions);
        persistEvents(events);

        db.getTransaction().commit();
        System.out.println("Db initialized");
    }
    catch (Exception e){
        e.printStackTrace();
    }
}

private List<Event> createEvents(List<String> eventNames, int year, int month) {
    List<Event> events = new ArrayList<>();

    for(int i = 1; i < eventNames.size()+1; i++){
        events.add(getIndividualEvent(eventNames, year, month, i));
    }

    return events;
}
```

```
private Event getIndividualEvent(List<String> eventNames, int year, int month, int i) {
   Event event = null;
   if(i <= 10){
     event = new Event(i, eventNames.get(i), UtilDate.newDate(year, month, 17));
   }
   if(i >= 11 && i <= 16){
     event = new Event(i, eventNames.get(i), UtilDate.newDate(year, month, 1));
   }
   if(i >= 17 && i <= 20){
     event = new Event(i, eventNames.get(i), UtilDate.newDate(year, month +1, 28));
   }
   return event;
}

private List<Question> createQuestionsByLanguaje(List<String>[] questionNames, List<Event>
events) {
   switch (Locale.getDefault().toString()){
     case "es_ES":
       return createQuestion(questionNames[0], events);
     case "en_US":
       return createQuestion(questionNames[1], events);
     case "eu_ES":
       return createQuestion(questionNames[2], events);
     default:
       return createQuestion(questionNames[0], events);
   }
}

private List<Question> createQuestion(List<String> questionNames, List<Event> events){
   List<Question> questions = new ArrayList<>();

   questions.add(events.get(0).addQuestion(questionNames.get(0), 1));
   questions.add(events.get(0).addQuestion(questionNames.get(1), 2));
   questions.add(events.get(10).addQuestion(questionNames.get(0), 1));
   questions.add(events.get(10).addQuestion(questionNames.get(2), 2));
   questions.add(events.get(16).addQuestion(questionNames.get(0), 1));
   questions.add(events.get(16).addQuestion(questionNames.get(3), 2));

   return questions;
}

private static void persistQuestions(List<Question> questions) {
   for(Question q: questions){
     db.persist(q);
   }
}

private static void persistEvents(List<Event> events) {
   for(Event ev: events){
```

```
        db.persist(ev);
    }
}
```

Como se puede observar no queda ningún bucle ni condicional en la función original y no hay ninguna función con más de 4 ramificaciones, haciendo todo más fácil de entender.

Refactor4(Capítulo5):

Como podemos ver esta función de DataAccess tiene 6 argumentos:

```
public User createUser(String username, String passwd, String dni, String name, String apellido,
boolean isAdmin) throws UserAlreadyExist {
    System.out.println(">> DataAccess: createUser => username="+username+" dni="+dni+"
name="+name+" apellido="+apellido+" isAdmin="+isAdmin);
        User user = db.find(User.class, dni);
        if (user==null ) {
            db.getTransaction().begin();
            user = new User(username, passwd, dni, name, apellido, isAdmin);
            db.persist(user);
            db.getTransaction().commit();
        }else {
            System.err.println(">> DataAccess: createUser => error UserAlreadyExist: "+
user.toString() + " already exists!");
            throw new UserAlreadyExist(user.toString() + " already exists!");
        }
        return user;
}
```

Esto se podría evitar convirtiéndolas en instancia de User antes de llamar a la función:

```
public User createUserInDB(User user) throws UserAlreadyExist {
    System.out.println(">> DataAccess: createUser => username="+user.getUsername()+"
dni="+user.getDni()+" name="+user.getName()
            +" apellido="+user.getApellido()+" isAdmin="+user.isAdmin());
        User userInDB = db.find(User.class, user.getDni());
        if (userInDB==null ) {
            db.getTransaction().begin();
            userInDB = new User(user.getUsername(), user.getPasswd(), user.getDni(),
user.getName(), user.getApellido(), user.isAdmin());
            db.persist(userInDB);
            db.getTransaction().commit();
        }else {
            System.err.println(">> DataAccess: createUser => error UserAlreadyExist: "+
userInDB.toString() + " already exists!");
            throw new UserAlreadyExist(userInDB.toString() + " already exists!");
        }
```

```
    return userInDB;
}
```

Además, he refactorizado el nombre ya que la función ya no crea un usuario sino que lo crea en la base de datos.

Alex Rivas Machín:

Refactorización 1(Capítulo 2):

- El método createForecast tiene muchas líneas de código, en concreto 22, por tanto, la vamos a dividir en distintas funciones:

```
public Forecast createForecast(String description, float gain, int questionNumber) throws
ForecastAlreadyExist, QuestionDoesntExist, DescriptionDoesntExist {
    System.out.println(">> DataAccess: createForecast => description="+description+"
gain="+gain+" Question="+questionNumber);
    if(description.isEmpty()) {
      System.err.println(">> DataAccess: createForecast => error DescriptionDoesntExist: La
descripción no puede estar vacía");
      throw new DescriptionDoesntExist("La descripción no puede estar vacía");
    }
    Question ques = db.find(Question.class, questionNumber);
    if (ques == null) {
     System.err.println(">> DataAccess: createForecast => error QuestionDoesntExist: No existe
una pregunta con este identificador: " + questionNumber);
     throw new QuestionDoesntExist("No existe una pregunta con este identificador: " +
questionNumber);
    }
    if (ques.DoesForecastExists(description)) {
      System.err.println(">> DataAccess: createForecast => error ForecastAlreadyExist: Esta
predicción ya existe");
      throw new ForecastAlreadyExist("Esta predicción ya existe");
    }
     db.getTransaction().begin();
    Forecast f = ques.addForecast(description, gain, ques);
    //db.persist(f);
    db.persist(ques); // db.persist(f) not required when CascadeType.PERSIST is added in
questions property of Event class
            // @OneToMany(fetch=FetchType.EAGER, cascade=CascadeType.PERSIST)
    db.getTransaction().commit();
    return f;
  }
```

- Como se puede ver los errores se tratan en el mismo método, uno a uno lo que hace que haya muchas líneas de código, he dividido estas llamadas en vari

Refactorización:

```
public Forecast createForecast(String description, float gain, int questionNumber) throws ForecastAlreadyExist, QuestionDoesntExist, DescriptionDoesntExist {
      System.out.println(">> DataAccess: createForecast => description="+description+" gain="+gain+" Question="+questionNumber);
      descriptionIsEmpty(description);
      Question ques = db.find(Question.class, questionNumber);
      invalidQuestion(ques, questionNumber);
      existingForecast(description, ques);
      db.getTransaction().begin();
      Forecast f = ques.addForecast(description, gain, ques);
      db.persist(ques);
      db.getTransaction().commit();
      return f;
}

1 usage  new *
public void descriptionIsEmpty(String description) throws DescriptionDoesntExist {
    if(description.isEmpty()) {
        System.err.println(">> DataAccess: createForecast => error DescriptionDoesntExist: La descripción no puede estar vacía");
        throw new DescriptionDoesntExist( s: "La descripción no puede estar vacía");
    }
}
1 usage  gomezbc *
public void invalidQuestion(Question ques, int questionNumber) throws QuestionDoesntExist, ForecastAlreadyExist {
    if (ques == null) {
        System.err.println(">> DataAccess: createForecast => error QuestionDoesntExist: No existe una pregunta con este identificador: " + questionNumber);
        throw new QuestionDoesntExist( s: "No existe una pregunta con este identificador: " + questionNumber);
    }
}
1 usage  new *
public void existingForecast(String description,Question ques) throws ForecastAlreadyExist {
    if (ques.DoesForecastExists(description)) {
        System.err.println(">> DataAccess: createForecast => error ForecastAlreadyExist: Esta predicción ya existe");
        throw new ForecastAlreadyExist( s: "Esta predicción ya existe");
    }
}
```

Refactorización (Capítulo 3):

- Este método se puede beneficiar de una simplificación, se va hacer con un bucle:

```
private List<Question> createQuestion(List<String> questionNames, List<Event> events){
    List<Question> questions = new ArrayList<>();

    questions.add(events.get(0).addQuestion(questionNames.get(0),  betMinimum: 1));
    questions.add(events.get(0).addQuestion(questionNames.get(1),  betMinimum: 2));
    questions.add(events.get(10).addQuestion(questionNames.get(0),  betMinimum: 1));
    questions.add(events.get(10).addQuestion(questionNames.get(2),  betMinimum: 2));
    questions.add(events.get(16).addQuestion(questionNames.get(0),  betMinimum: 1));
    questions.add(events.get(16).addQuestion(questionNames.get(3),  betMinimum: 2));

    return questions;
}
```

- Como es corto aquí otro:

```
private List<Question> createQuestionsByLanguaje(List<String>[] questionNames, List<Event> events) {
    switch (Locale.getDefault().toString()){
        case "es_ES":
            return createQuestion(questionNames[0], events);
        case "en_US":
            return createQuestion(questionNames[1], events);
        case "eu_ES":
            return createQuestion(questionNames[2], events);
        default:
            return createQuestion(questionNames[0], events);
    }
}
```

Refactorización:

```java
private List<Question> createQuestion(List<String> questionNames, List<Event> events){
    List<Question> questions = new ArrayList<>();
    for (Event event : events) {
        for (int i = 0; i < questionNames.size(); i++) {
            String questionName = questionNames.get(i);
            int betMinimum = i + 1; // Otra lógica si es necesario

            Question question = event.addQuestion(questionName, betMinimum);
            questions.add(question);
        }
    }
    return questions;
}
```

```java
private List<Question> createQuestionsByLanguage(List<String>[] questionNames, List<Event> events) {
    Map<String, List<String>> languageQuestionMap = new HashMap<>();
    languageQuestionMap.put("es_ES", questionNames[0]);
    languageQuestionMap.put("en_US", questionNames[1]);
    languageQuestionMap.put("eu_ES", questionNames[2]);

    String defaultLanguage = "es_ES";
    List<String> selectedQuestions = languageQuestionMap.getOrDefault(Locale.getDefault().toString(), languageQuestionMap.get(defaultLanguage));

    return createQuestion(selectedQuestions, events);
}
```

Refactorización 3 (Capítulo 4):

- Tenemos alguna pregunta de código que se duplica:

```java
List<Question> questions = createQuestionsByLanguaje(questionNames, events);

questions.get(0).addForecast( description: "Athletic",  gain: 1.5f ,questions.get(0));
questions.get(0).addForecast( description: "Atlético",  gain: 1.4f ,questions.get(0));
questions.get(1).addForecast( description: "Athletic",  gain: 1.8f ,questions.get(1));
questions.get(1).addForecast( description: "Atlético",  gain: 1.2f ,questions.get(1));
```

Refactorización:

```
76          List<Question> questions = createQuestionsByLanguaje(questionNames, events);
77
78          getQuesForecast(questions);
79          persistQuestions(questions);
80          persistEvents(events);
81
82          db.getTransaction().commit();
83          System.out.println("Db initialized");
84      }
85      catch (Exception e){
86          e.printStackTrace();
87      }
88  }
89
        1 usage  new *
90  @   private void getQuesForecast(List<Question> questions) {
91          // Create and add forecasts to questions
92          Question question1 = questions.get(0);
93          question1.addForecast( description: "Athletic",  gain: 1.5f, question1);
94          question1.addForecast( description: "Atlético",  gain: 1.4f, question1);
95
96          Question question2 = questions.get(1);
97          question2.addForecast( description: "Athletic",  gain: 1.8f, question2);
98          question2.addForecast( description: "Atlético",  gain: 1.2f, question2);
99      }
```

Refactorización 1 ("Write short units of code")

Código inicial

```
public Bet createBet (String dni, float betMoney, int forecastNumber) throws
BetAlreadyExist, UserDoesntExist, ForecastDoesntExist {
    System.out.println(">> DataAccess: createBet => user=" + dni + " dinero
apostado="+betMoney + " al forecast=" + forecastNumber);
    Forecast forecast = db.find(Forecast.class, forecastNumber);
    User u = db.find(User.class, dni);
    if(u==null) {
       System.err.println(">> DataAccess: createBet => error UserDoesntExist: No
hay un usuario con este DNI en la base de datos, dni="+dni);
       throw new UserDoesntExist("No hay un usuario con este DNI en la base de
datos, dni="+dni);
    }
    if (forecast == null) {
       System.err.println(">> DataAccess: createBet => error ForecastDoesntExist:
No hay un pronostico con este identificador en la base de datos,
forecastNumber="+forecastNumber);
       throw new ForecastDoesntExist("No hay un pronostico con este identificador
en la base de datos, forecastNumber="+forecastNumber);
    }

    if ( u.DoesBetExists(forecastNumber) != null) {
       System.err.println(">> DataAccess: createBet => error BetAlreadyExist: Ya
existe una apuesta a este pronostico");
       throw new BetAlreadyExist("Ya existe una apuesta a este pronostico");
    }
       db.getTransaction().begin();
           Bet b = u.addBet(betMoney, forecast);
           db.persist(u);
          db.getTransaction().commit();
          return b;
}
```

Código refactorizado

```
public Bet createBet (String dni, float betMoney, int forecastNumber) throws
BetAlreadyExist, UserDoesntExist, ForecastDoesntExist {
    System.out.println(">> DataAccess: createBet => user=" + dni + " dinero
apostado="+betMoney + " al forecast=" + forecastNumber);

    User u = getUserFromDB(dni);
    Forecast forecast = getForecastFromDB(forecastNumber);
    doesBetAlreadyExist(forecastNumber, u);

    db.getTransaction().begin();
        Bet b = u.addBet(betMoney, forecast);
        db.persist(u);
       db.getTransaction().commit();
    return b;
}

private static void doesBetAlreadyExist(int forecastNumber, User u) throws
BetAlreadyExist {
    if ( u.DoesBetExists(forecastNumber) != null) {
       System.err.println(">> DataAccess: createBet => error BetAlreadyExist: Ya
existe una apuesta a este pronostico");
       throw new BetAlreadyExist("Ya existe una apuesta a este pronostico");
    }
}

private static Forecast getForecastFromDB(int forecastNumber) throws
ForecastDoesntExist {
    Forecast forecast = db.find(Forecast.class, forecastNumber);
    if (forecast == null) {
       System.err.println(">> DataAccess: createBet => error ForecastDoesntExist:
No hay un pronostico con este identificador en la base de datos, forecastNumber="+
forecastNumber);
       throw new ForecastDoesntExist("No hay un pronostico con este identificador
en la base de datos, forecastNumber="+ forecastNumber);
    }
    return forecast;
}

private static User getUserFromDB(String dni) throws UserDoesntExist {
    User u = db.find(User.class, dni);
    if(u==null) {
       System.err.println(">> DataAccess: createBet => error UserDoesntExist: No
hay un usuario con este DNI en la base de datos, dni="+ dni);
       throw new UserDoesntExist("No hay un usuario con este DNI en la base de
datos, dni="+ dni);
    }
    return u;
}
```

Descripción del error concreto detectado y descripción de la refactorización realizada.

El método createBet tiene 20 líneas, por lo que supera el límite de 15 que sugiere el libro.

Para solucionarlo, como sugiere el libro, dividiremos las líneas en distintos métodos agrupándolas por la acción que realizan.

Miembro que ha realizado la refactorización

Borja Gómez

Refactorización 2 ("Write simple units of code")

En la clase MainGUI.java

Código inicial

```
private JButton getJButtonLogin() {
    if (jButtonLogin == null) {
       jButtonLogin = new
JButton(ResourceBundle.getBundle("Etiquetas").getString("MainGUI.jButtonLogin.text"
)); //$NON-NLS-1$ //$NON-NLS-2$
       jButtonLogin.setBounds(20, 170, 140, 32);
       jButtonLogin.setBackground(new Color(255, 255, 255));
       ImageIcon icon = new
ImageIcon(EventInfoPanel.class.getResource("/icons/user-login.png"));
       Image scaledIcon = icon.getImage().getScaledInstance(20, 20,
Image.SCALE_SMOOTH);
       jButtonLogin.setIcon(new ImageIcon(scaledIcon));
       jButtonLogin.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e1) {
             BLFacade facade = MainGUI.getBusinessLogic();
             if(userRegistered!= null) {
                userError.setText("<html>Cierra sesión antes de iniciar <br>sesión
con otro usuario</html>");
                userError.setVisible(true);
             }else {
                User user = null;
                try {
                   user = facade.getUser(textField.getText());
                   if(!user.checkCredentials(new
String(pwdIngreseSuContrasea.getPassword()))) userError.setText("La contraseña es
incorrecta");
                   else {
                         setUserRegistered(user);
                         userError.setVisible(false);
                      if(user.isAdmin()) {
                         JFrame a = new AdminGUI();
                         a.setVisible(true);
                      }else {
                         JFrame a = new UserGUI();
                         a.setVisible(true);
                      }
                   }
                }catch(UserDoesntExist e2) {
                   userError.setVisible(true);
                   userError.setText(e2.getMessage());
                }
             }

          }
       });
    }
    return jButtonLogin;
}
```

Código refactorizado

```
private JButton getJButtonLogin() {
    if (jButtonLogin == null) {
       jButtonLogin = new
JButton(ResourceBundle.getBundle("Etiquetas").getString("MainGUI.jButtonLogin.text"
)); //$NON-NLS-1$ //$NON-NLS-2$
       jButtonLogin.setBounds(20, 170, 140, 32);
       jButtonLogin.setBackground(new Color(255, 255, 255));
       ImageIcon icon = new
ImageIcon(EventInfoPanel.class.getResource("/icons/user-login.png"));
       Image scaledIcon = icon.getImage().getScaledInstance(20, 20,
Image.SCALE_SMOOTH);
       jButtonLogin.setIcon(new ImageIcon(scaledIcon));
       jButtonLogin.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e1) {
```

```
            BLFacade facade = MainGUI.getBusinessLogic();
            validateLogin(facade);

        }

        private void validateLogin(BLFacade facade) {
            if(userRegistered!= null) {
                userError.setText("<html>Cierra sesión antes de iniciar <br>sesión
con otro usuario</html>");
                userError.setVisible(true);
            }else {
                User user = null;
                try {
                    user = facade.getUser(textField.getText());
                    checkUserCredentials(user);
                }catch(UserDoesntExist e2) {
                    userError.setVisible(true);
                    userError.setText(e2.getMessage());
                }
            }
        }

        private void checkUserCredentials(User user) {
            if(!user.checkCredentials(new
String(pwdIngreseSuContrasea.getPassword()))) userError.setText("La contraseña es
incorrecta");
            else {
                    setUserRegistered(user);
                    userError.setVisible(false);
                    showGUIBasedOnRole(user);
            }
        }

        private void showGUIBasedOnRole(User user) {
            if(user.isAdmin()) {
                JFrame a = new AdminGUI();
                a.setVisible(true);
            }else {
                JFrame a = new UserGUI();
                a.setVisible(true);
            }
        }
    });
  }
  return jButtonLogin;
}
```

Descripción del error concreto detectado y descripción de la refactorización realizada.

Al principio tenía una complejidad cognitiva de 20, superando los 15 de máximo que sugiere el capítulo. Por lo que hemos separado varios if a métodos externos, dentro del actionListener del botón, hemos dividido estos if en métodos dependiendo de su función. Así ha quedado la estructura:

- actionPerformed
  - validateLogin
    - checkUserCredentials
      - showGUIBasedOnRole

Hemos conseguido que el código sea más sencillo de leer y entender su funcionamiento.

Miembro que ha realizado la refactorización

Borja Gómez

Refactorización 3 ("Duplicate code")

En la clase ListUsersGUI.java

Código inicial

```
tabbedPane.setFont(new Font("Roboto", Font.PLAIN, 14));
scrollPaneUser.setFont(new Font("Roboto", Font.PLAIN, 14));
scrollPaneAdmin.setFont(new Font("Roboto", Font.PLAIN, 14));
btnEliminarUsuario.setFont(new Font("Roboto", Font.BOLD, 14));
```

Código refactorizado

```
public static final String tabbedPaneFont = "Roboto";


tabbedPane.setFont(new Font(tabbedPaneFont, Font.PLAIN, 14));
scrollPaneUser.setFont(new Font(tabbedPaneFont, Font.PLAIN, 14));
scrollPaneAdmin.setFont(new Font(tabbedPaneFont, Font.PLAIN, 14));
btnEliminarUsuario.setFont(new Font(tabbedPaneFont, Font.BOLD, 14));
```

Descripción del error concreto detectado y descripción de la refactorización realizada.

Moviendo el texto repetido a una constante evitamos que al modificarlo en distintas partes del código la fuente de todo el panel no quede irregular si se nos olvida cambiarlo en alguna parte.

Miembro que ha realizado la refactorización

Borja Gómez

Refactorización 4 ("Keep unit interfaces small")

Clase BLFacadeImplementation.java

Código inicial

```
@WebMethod
public User createUser(String username, String passwd, String dni, String
name, String apellido, boolean isAdmin) throws UserAlreadyExist {
 dbManager.open(false);
 User user = null;
 try {
  user = dbManager.createUserInDB(new User(username, passwd, dni, name,
apellido, isAdmin));
 }catch(UserAlreadyExist e) {
  throw e;
 }finally {
  dbManager.close();
 }
 return user;
}
```

In CreateUserGUI.java file:

```
facade.createUser(username.getText(), new String(passwd.getPassword()),
dni.getText(), name.getText(), surname.getText(),
checkbxisAdmin.isSelected());
```

Código refactorizado

```
@WebMethod
public User createUser(User user) throws UserAlreadyExist {
 dbManager.open(false);
 try {
  user = dbManager.createUserInDB(user);
 }catch(UserAlreadyExist e) {
  throw e;
 }finally {
  dbManager.close();
 }
 return null;
}
```

In CreateUserGUI.java file:

```
User user = new User(username.getText(), new String(passwd.getPassword()),
dni.getText(), name.getText(), surname.getText(),
checkbxisAdmin.isSelected());
facade.createUser(user);
```

Descripción del error concreto detectado y descripción de la refactorización realizada.

Aunque ya hubiésemos refactorizado el método createUser en DataAccess para que no usase más de 4 parámetros, el método en BLFacade no habíamos arreglado esta mala práctica.

Solamente hemos tenido que modificar CreateUserGui para que sea adapte a este cambio.

Miembro que ha realizado la refactorización

Borja Gómez